ARMY RESEARCH LABORATORY

The Design Constraints for the Memory Systems of Useful SMPs

by D. M. Pressel

ARL-TR-2146 January 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Abstract

In recent years, many vendors have produced *cache coherent shared memory symmetric multiprocessors*. While most of the systems that used, at most, eight processors have been successes, the same statement cannot be made for the larger, more scalable systems. Some of the larger systems have been extremely successful, others have been marginally to reasonably successful, and a few have been outright failures. Based on the author's experience programming the KSR1, Convex Exemplar, Silicon Graphics Inc. (SGI) Challenge and Power Challenge, and the SGI Origin 2000, some insights into key design issues for a successful *cache coherent shared memory symmetric multiprocessor* are discussed. The report concludes with a frequently overlooked issue—the cost effectiveness of some of these designs. In particular, any design that requires the widespread replication of key data structures will have a hard time establishing its cost effectiveness (even if it does meet the requirements for performance and scalability).

Table of Contents

INTENTIONALLY LEFT BLANK.

1. Introduction

Traditionally, the list of commercially available computers could be separated into the following groups:

- Uniprocessors.
- **SMPs** based on a small number (2–8) of powerful processors.
- SMPs based on a moderate number (no more than 64 and frequently less than 20) of weak microprocessors.
- **MPPs** based on a large number (hundreds to thousands) of weak microprocessors.

There have been an increasing number of attempts at producing more powerful SMPs. The following is a list of some of these efforts:

- Use larger numbers of vector processors (e.g., Japanese Numerical Wind Tunnel—currently 160 processors rated at 1.6 GFLOPS each).
- Use more powerful **RISC** processors (e.g., Silicon Graphics Inc. [SGI] Power Challenge).
- Use larger numbers of processors (e.g., KSR1 and Convex Exemplar).

Note: This work was made possible through a grant of computer time by the Department of Defense (DOD) High Performance Computing Modernization Program. Additionally, it was funded as part of the Common High Performance Computing Software Support Initiative administered by the DOD High Performance Computing Modernization Program.

Note: All items in bold type are defined in the Glossary.

- Use larger numbers of more powerful RISC processors (e.g., SGI Origin 2000 and SUN HPC 10000).

This report restricts itself to issues involving creating a scalable shared memory SMP from RISC (or **CISC**) processors.

The basis for this report is the author's experience programming on the KSR1, SGI Challenge and Power Challenge, Convex Exemplar SPP-1600, and the SGI Origin 2000. This experience includes a combination of small test programs designed to measure specific features in the system and the porting and parallelization of a complete production scientific code (F3D) to these systems (with the exception of the KSR1, which was already turned off when this effort began). What made this effort particularly noteworthy is that F3D is an implicit **CFD** code and, at the start of this project, it was already known to perform quite poorly on RISC-based architectures. Furthermore, many of the author's colleagues doubted that it would ever perform well on any architecture that used a memory hierarchy. Finally, it was common knowledge among experts in the field that implicit CFD codes could not be parallelized without adversely affecting their results/efficiency (fortunately, no one bothered to tell this to the author).

The author (in conjunction with J. Sahu, K. R. Heavey, and others from the U.S. Army Research Laboratory [ARL]) successfully demonstrated that, in fact, F3D could be ported to and parallelized for some RISC-based SMPs (Collins et al. 1997; Pressel 1997, 1999; Sahu et al. 1988; Sturek, Tezduyar, and Muzio, to be published).

Based on the results from this work, it is explained why it is important to maintain as high a bisectional memory bandwidth as possible. In particular, architectures that rely upon an extremely large per node cache (e.g., COMA architectures like the KSR1 or the CTI cache in the Convex Exemplar) as a method for tolerating low bisectional memory bandwidths and or extremely high levels of off-node memory latency will not perform well when running codes such as F3D. This is an important observation since this class of codes represents a natural constituency for shared

memory SMPs, while many other classes of codes will run just as well on an MPP. Therefore, unless a scalable shared memory SMP does a good job of supporting codes such as F3D, it may be hard to justify the added expense and limited scalability normally associated with shared memory systems.

Finally, the costs associated with replicating data are reviewed. There are many ways in which this might happen (e.g., large per node caches, explicitly copying data into local memory, or replicating the data on a per process basis for message-passing codes). The key point here is that, no matter how this occurs, the replication of data will decrease the maximum job size that can be run, while increasing the cost of running an individual job. Therefore, any system that relies upon this strategy will have trouble proving its cost effectiveness, unless the strategy results in a major boost in performance.

2. What Makes Implicit CFD Codes So Hard to Parallelize?

In order to understand why shared memory SMPs are inherently well suited for running parallelized implicit CFD codes, one needs to understand something about how these codes work. Depending on the nature of the problem and the algorithm used to solve it, one can classify many codes into one of three categories on the basis of the communication patterns:

1) Particles or grid points only interact with their nearest neighbors. In this case, one can separately store the values from the previous time step and the current time step. This makes the calculation of the values for the current time step independent of each other and results in a highly parallelizable program.

2) Particles or grid points may naturally form clusters (e.g., stars forming a galaxy will only weakly interact with other galaxies). In this case, one can separately calculate the values associated with each cluster; although, within a cluster, the amount of parallelism may be highly limited. Usually, this is an approximation, but, under ideal conditions, it can result

in a significant amount of parallelism without a significant decrease in the accuracy of the results.

3) Particles or grid points may be grouped in a very small number of clusters or zones (possibly just one). While, in theory, it may be possible to process the clusters or zones in parallel, this is likely to result in significant problems relating to load balancing. The two most appropriate solutions to this problem are as follows:

   a) Parallelize the processing of individual clusters or zones and then process the clusters or zones one at a time. Depending on the algorithm, this may support only a modest level of parallelism.

   b) Split the clusters or zones into many smaller clusters or zones (a process known as domain decomposition). The problem here is that some codes (e.g., implicit CFD codes like F3D) propagate information throughout a zone in a single time step. For example, if a hammer hits one side of the zone, then the entire zone will feel the shock wave in a single time step. If the zone is split into a large number of small pieces, this behavior is lost and the run may fail to converge to a solution. Alternatively, one may have to significantly decrease the size of the time step to avoid the convergence problems. A third alternative is to change the algorithm, but this choice is, in general, not well received by the computational scientists!

If one considers case 3a in greater detail and, in particular, considers how implicit CFD codes behave, some important patterns become apparent:

- Some loops have dependencies in them in one or even two directions.

- In general, there will be two or more loops with incompatible dependencies, which prevent one from parallelizing all of the loops under a single outer loop.

- Historically, these codes have been considered to be good performers on vector processors. This guarantees that, for most if not all of the loops, they are, in theory, parallelizable in at least one direction.

If one looks further at what it takes to turn vectorizable code into parallel code, the following observations come to mind:

- From a software perspective, one needs to interchange loops so that the parallelizable loop has as much work associated with it as possible.

- Also, from a software perspective, some of the loops will have so little work associated with them that it is hard to justify the overhead associated with parallelizing them. However, on a distributed memory message-passing environment, there is no other option.

- From a hardware perspective, since different loops are likely to be parallelized in different directions, attempts to parallelize this code in a distributed memory message-passing environment will require frequent data redistributions. The most natural way to carry this out involves sending huge numbers of small messages, which results in a code that is strongly limited by the latency of interprocessor communication. Even if one can cluster messages together so that latency is less of a problem, the aggregate bandwidth for interprocessor communication may become a problem.

- On the other hand, if one considers the possibility of using a shared memory SMP, one sees that it is no longer necessary to parallelize the small loops in the boundary condition routines. This dramatically reduces the need for explicitly choreographed data motion.

- Furthermore, for the remaining places where data redistributions (now called matrix transposes) are still desirable, it should be noted that they can now be performed at the full

speed of the memory system, which is almost always much greater than the aggregate bandwidth for interprocessor communication on the average MPP.

- Finally, it is this author's belief that, on a shared memory system, one is generally more likely to be able to store multiple copies of key areas (e.g., the array and its transpose) for the complete length of a run. When dealing with relatively invariant arrays, this can be a particularly useful way to reduce the requirement for data redistribution by as much as an order of magnitude.

3. The Natural Constituency for Shared Memory Architectures

The obvious question when discussing the need for scalable SMPs is to ask: Why are they needed at all? Until that question has been answered, one may have trouble identifying the necessary characteristics for a successful scalable SMP. Clearly, most current parallel programs run just fine on the MPPs they were written for. Therefore, one should look at the programs that perform poorly on most MPPs and those that were considered to be nonparallelizable in the first place.

There are any number of reasons why a program might perform poorly on an MPP (e.g., too many small messages, too many cache misses, etc.). Many **HPF** programs fall into this category, as do some programs that make extensive use of collective communications (e.g., data redistributions, reductions). In many cases, experience has shown that these codes perform best on scalable SMPs that have well-implemented **MPI** libraries (e.g., threads based, which support a very low latency and can minimize the amount of unnecessary memory traffic).

In the case of programs that are considered to be nonparallelizable on traditional MPPs (e.g., F3D), efficient support for compiler directive-based loop-level parallelism seems to be the key. Additionally, there is a strong benefit for an efficient implementation of shared memory, so that one

can parallelize the code incrementally (with a high probability that some boundary condition routines will never be parallelized).

4. What Are the Special Hardware Requirements of These Codes?

While one can argue things all day long, this author believes that the two main requirements are as follows:

1) Since shared memory SMPs will almost always have a greater memory latency than their MPP cousins, they have a strong need for large external caches (e.g., 1–8 MB) that can sharply decrease the cache miss rate (preferably with long cache line sizes [e.g., 128–1,024 B]).

2) The upper bound on the effective cost of a cache miss that misses all the way back to main memory must be kept to a minimum. This must be the case under as wide a range of conditions as possible. This implies the need for a high bisectional memory bandwidth, as well as a low upper bound on the cost of the cache miss. Only in that way can one be certain that delays due to insufficient bisectional memory bandwidth will not dwarf the cost of the cache miss itself. Unfortunately, both the KSR1 and the Convex Exemplar SPP-1600 have shown problems in this area.

Both the KSR1 and the Convex Exemplar have attempted to use large **DRAM** caches to avoid these problems. Experience with F3D and similar programs has shown that, at least with programs parallelized using loop-level parallelism, the direction of parallelization will change too often for these techniques to be of much value. In some cases, they even seemed to be counterproductive. There is reason to believe that, for HPF programs as well as MPI-based programs making extensive use of collective communications, a similar statement might also apply.

On the Convex Exemplar, we also experimented with using local memory to maintain copies of key arrays (ones that were relatively invariant throughout the life of the run). While this helped to some extent, the benefits were limited. Furthermore, the cost of this approach, both in terms of the need for extra memory and in terms of the extra time required to make all of the copies, makes this approach undesirable and of questionable value in a production environment.

5. The Whys and Wherefores of Replicated Data Structures

If one looks at parallelized versions of ray-tracing codes and some chemistry codes, one discovers something interesting. Unlike the codes people are used to talking about, these codes are difficult to parallelize without replicating the entirety of all of the major data structures. It is not difficult to see how this could raise the cost of a system by one or more orders of magnitude (depending on how large a problem one intends to work on).

On the other hand, a scalable shared memory system would seem to have a natural advantage here. Only one copy of the major data structures needs to reside in memory. Unfortunately, there are some potential problems with this simplistic view.

- One can get bank conflicts. This can be an especially big problem on systems like the SGI Origin 2000 and the Convex Exemplar SPP-1600, where data are allocated to a node's memory banks a page at a time. On the other hand, systems such as the SUN HPC 10000 should have fewer of these problems since they manage things a cache line at a time.

- Some systems such as the KSR1 and the Convex Exemplar SPP-1600 will perform poorly if a disproportionately large number of cache misses go off node.

- If one attempts to make up for a systems shortcomings by using large DRAM caches (e.g., the KSR1 and the Convex Exemplar SPP-1600), then once again one is faced with the cost of replicating the major data structures in the DRAM cache for every node. While this might

simplify the programming, it can still result in excessive hardware costs (although using larger numbers of processors per node can help to mitigate these costs).

Therefore, even when the code does not explicitly replicate key data structures on every node, one needs to make sure that the hardware is not designed to do this behind the user's back. This is not to say that one should never replicate key data structures. If they are relatively invariant, then it may be desirable to store two or even three copies of key arrays, with different ordering of the indices. This can serve to greatly reduce the number of cache misses and/or the number of transpose operations that one needs to perform during the life of the run. The key difference here is that the number of copies of these data structures is a constant rather than being a function of the number of processors being used. As such, the amount of extra memory required is tightly bounded, as is the cost of that extra memory.

6. The Limitations of the Concept of a Scalable SMP

One final point is that people are used to dealing with MPPs that scale to hundreds or even thousands of processors. Therefore they assume that a successful scalable SMP needs the same level of scalability. To a certain extent, this is not a bad idea. After all, people want to run message-passing and even Cray T3D/T3E **SHMEM**-type codes on these machines. On the other hand, such scalability is not free. The larger a machine (any kind of parallel computer, not just an SMP), the harder it is to build it with an acceptable level of stability, reliability, and performance. Therefore, unless one's customer base is demanding very large systems, there can be a substantial amount of beauty to moderate-sized systems.

Furthermore, most jobs using HPF, loop-level parallelism, or needing the ultralow latency for message passing that shared memory SMPs tend to offer are generally not all that scalable. This author has heard statements referring to limits of, at most, 16 processors. While this author has done much better than that on the SGI Origin 2000, it is clear that for small- to moderate-sized jobs parallelized with loop-level parallelism, it probably is counterproductive to parallelize most of the

boundary condition routines. However, this raises the specter of **Amdahl's Law** and therefore makes it clear that, as the system size passes 100 processors, the law of diminishing returns will come into play. If one accepts that these classes of jobs represent the natural constituency for scalable SMPs, then one must also conclude that the incremental benefits from making SMPs with more than 100 processors is, at best, limited, and therefore one should only make such systems if the incremental costs are very small indeed.

7. Conclusion

It has been shown that the design of a scalable shared memory SMP is highly dependent on the design of the memory system. In particular, a high bisectional memory bandwidth is critical. Also relying on large DRAM caches will frequently be an unacceptable substitute for having a high bisectional memory bandwidth. Furthermore, the reliance of an architecture on the widespread replication of major data structures can either sharply limit the maximum job size and/or dramatically increase the system cost.

At the present time, the SGI Origin 2000 appears to be the most successful scalable shared memory SMP on the market, while the SUN E10000 and HPC10000 are probably the second runners up. For some markets, the SUN systems seem to be much more successful, even though they are less scalable. Until recently, one of the key drawbacks to the SUN systems was the lack of a 64-bit operating system for the E10000/HPC10000. While, for many applications, this did not matter, for shared memory applications parallelized using loop-level parallelism, this put an all too small upper bound on the problem sizes that could be run on this machine. It is unclear at this point in time how long it will take for the third-party software vendors to migrate to the 64-bit programming environment.

8. References

Collins, J. P., D. M. Pressel, C. J. Nietubicz, J. Sahu, K. Heavey, P. Weinacht, H. Edge, M. Behr, and J. Clarke. "ARL Zonal Navier-Stokes Solvers, CHSSI CFD-6 Project Annual Report, 1 April–30 September 1996." ARL-MR-364, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, August 1997.

Pressel, D. M. "Early Results From the Porting of the Computational Fluid Dynamics Code, F3D to the Silicon Graphics Power Challenge." ARL-TR-1562, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, December 1997.

Pressel, D. M. "Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Exemplar (SPP-1000 and SPP-1600)." ARL-TR-1923, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, March 1999.

Sahu, J., D. M. Pressel, K. R. Heavey, and C. J. Nietubicz. "Parallel Application of a Navier-Stokes Solver for Projectile Aerodynamics." Published in *Parallel Computational Fluid Dynamics, Recent Developments and Advances Using Parallel Computers* and *Proceedings of the Parallel CFD'97 Conference*, Manchester, UK, 19–21 May 1997. D. R. Emerson, J. Periaux, A. Ecer, N. Satofuka, and P. Fox (editors), Amsterdam: Elsevier, 1998.

Sturek, W. B., T. E. Tezduyar, and P. Muzio. "The Army High Performance Computing Research Center - A Unique Resource for Defense Basic Research and Education." U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, to be published.

INTENTIONALLY LEFT BLANK.

Glossary

Amdahl's Law: As one scales fixed-sized problems to large numbers of processors, the percentage of serial work (nonparallelized work) will come to dominate the run time, thereby placing an upper bound on the speedup one can achieve through parallelization.

CFD: Computational Fluid Dynamics.

CISC: Complicated Instruction Set Computer.

DRAM: Slower, cheap memory used as the main memory inmost computers.

HPF: High Performance Fortran.

MPI: Message Passing Interface.

MPP: Massively Parallel Processor.

RISC: Reduced Instruction Set Computer.

SHEM: "Shared memory," an approach to low latency message passing pioneered by Cray Research.

SMP: Symmetric Multiprocessor.

SRAM: Fast, expensive memory used in caches and as the main memory of some high-end and special-use computers.

INTENTIONALLY LEFT BLANK.

NO. OF COPIES ORGANIZATION

2 DEFENSE TECHNICAL INFORMATION CENTER
DTIC DDA
8725 JOHN J KINGMAN RD
STE 0944
FT BELVOIR VA 22060-6218

1 HQDA
DAMO FDQ
D SCHMIDT
400 ARMY PENTAGON
WASHINGTON DC 20310-0460

1 OSD
OUSD(A&T)/ODDDR&E(R)
R J TREW
THE PENTAGON
WASHINGTON DC 20301-7100

1 DPTY CG FOR RDA
US ARMY MATERIEL CMD
AMCRDA
5001 EISENHOWER AVE
ALEXANDRIA VA 22333-0001

1 INST FOR ADVNCD TCHNLGY
THE UNIV OF TEXAS AT AUSTIN
PO BOX 202797
AUSTIN TX 78720-2797

1 DARPA
B KASPAR
3701 N FAIRFAX DR
ARLINGTON VA 22203-1714

1 NAVAL SURFACE WARFARE CTR
CODE B07 J PENNELLA
17320 DAHLGREN RD
BLDG 1470 RM 1101
DAHLGREN VA 22448-5100

1 US MILITARY ACADEMY
MATH SCI CTR OF EXCELLENCE
DEPT OF MATHEMATICAL SCI
MADN MATH
THAYER HALL
WEST POINT NY 10996-1786

NO. OF COPIES ORGANIZATION

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL DD
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RESEARCH LAB
AMSRL CS AS (RECORDS MGMT)
2800 POWDER MILL RD
ADELPHI MD 20783-1145

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRL CI LL
2800 POWDER MILL RD
ADELPHI MD 20783-1145

ABERDEEN PROVING GROUND

4 DIR USARL
AMSRL CI LP (BLDG 305)

NO. OF COPIES	ORGANIZATION
1	PM CHSSI JOHN GROSH SUITE 510 1010 N GLEBE ROAD ARLINGTON VA 22201
1	RICE UNIVERSITY MCHNCL ENGRNG AND MTRLS SCI MAREK BEHR MS 321 6100 MAIN STREET HOUSTIN TX 77005
1	COMMANDER CODE C2892 CLINT HOUSH 1 ADMINISTRATION CIR CHINA LAKE CA 93555
2	WL FIMC STEPHEN SCHERR BILL STRANG BLDG 450 2645 FIFTH ST SUITE 7 WPAFB OH 45433-7913
1	NSWC A B WARDLAW CODE B44 SILVER SPRING MD 20903-5640
1	NAVAL RSRCH LAB CODE 6400 JAY BORIS 4555 OVERLOOK AVE SW WASHINGTON DC 20375-5344
1	NAVAL RSRCH LAB CODE 6410 RAVI RAMAMURTI WASHINGTON DC 20375-5344
1	ARMY AEROFLIGHT DYNAMICS DIRECTORATE ROBERT MEAKIN MS 258 1 MOFFETT FIELD CA 94035-1000
1	NAVAL RSRCH LAB CODE 7320 J W MCCAFFREY JR HEAD OCEAN DYNAMICS AND PREDICTION BRANCH STENNIS SPACE CENTER MS 39529

NO. OF COPIES	ORGANIZATION
1	NAVAL RSRCH LAB GEORGE HEBURN RSRCH OCEANOGRAPHER CNMOC BLDG 1020 RM 178 STENNIS SPACE CENTER MS 39529
1	US AIR FORCE WRIGHT LAB WL FIM JOSEPH J S SHANG 2645 FIFTH STREET STE 6 WPAFB OH 45433-7912
1	USAF PHILIPS LAB OLAC PL RKFE CPT SCOTT G WIERSCHKE 10 EAST SATURN BLVD EDWARDS AFB CA 93524-7680
1	USAE WATERWAYS EXPERIMENT STATION CEWES HV C JEFFREY P HOLLAND 3909 HALLS FERRY ROAD VICKSBURG MS 39180-6199
1	US ARMY CECOM RD&E CTR AMSEL RD C2 BARRY S PERLMAN FT MONMOUTH NJ 07703
1	SPAWARSYSCEN (D4402) ROBERT A WASILAUSKY BLDG 33 RM 0071A 53560 HULL ST SAN DIEGO CA 92152-5001
1	US AIR FORCE RESEARCH LAB INFORMATION DIRECTORATE RICHARD W LINDERMAN 26 ELECTRONIC PARKWAY ROME NY 13441-4514
1	US AIR FORCE RESEARCH LAB PROPULSION DIRECTORATE LESLIE PERKINS 5 POLLUX DR EDWARDS AFB CA 93524-7048
1	AIR FORCE RESEARCH LAB/DEHE ROBERT PETERKIN 3550 ABERDEEN AVE SE KIRTLAND AFB NM 87117-5776

NO. OF COPIES	ORGANIZATION
1	SPACE & NAVAL WARFARE SYS CTR CODE D7305 KEITH BROMLEY BLDG 606 RM 325 53140 SYSTEMS ST SAN DIEGO CA 92152-5001
1	UNVRSTY OF MINNESOTA DEPT OF ASTRONOMY PROF P WOODWARD 356 PHYSICS BLDG 116 CHURCH STREET SE MINNEAPOLIS MN 55455
1	RICE UNIVERSITY MCHNCL ENGRNG AND MTRLS SCI TAYFUN TEZDUYAR DEPT CHRMN MS 321 6100 MAIN ST HOUSTON TX 77005
1	DIRECTOR ARMY HIGH PERFORMANCE COMPUTING RSRCH CTR BARBARA BRYAN 1200 WASHINGTON AVE SOUTH MINNEAPOLIS MN 55415
1	DIRECTOR ARMY HIGH PERFORMANCE COMPUTING RSRCH CTR GRAHAM V CANDLER 1200 WASHINGTON AVE SOUTH MINNEAPOLIS MN 55415
1	NAVAL CMND CONTROL AND OCEAN SURVEILLANCE CTR L PARNELL HPC CRDNTR & DIR NCCOSC RDTE DIV D3603 49590 LASSING ROAD SAN DIEGO CA 92152-6148

NO. OF COPIES	ORGANIZATION
	ABERDEEN PROVING GROUND
15	DIR USARL AMSRL CI N RADHAKRISHNAN AMSRL CI H C NIETUBICZ AMSRL CI HA W STUREK A MARK R NAMBURU AMSRL CI HC D PRESSEL D HISLEY C ZOLTANI A PRESSLEY T KENDALL P DYKSTRA AMSRL WM BC H EDGE J SAHU K HEAVEY P WEINACHT

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 2000	Final, Jan 96-Dec 96

4. TITLE AND SUBTITLE

The Design Constraints for the Memory Systems of Useful SMPs

5. FUNDING NUMBERS

9UHMCL

6. AUTHOR(S)

D. M. Pressel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-2146

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

In recent years, many vendors have produced *cache coherent shared memory symmetric multiprocessors*. While most of the systems that used, at most, eight processors have been successes, the same statement cannot be made for the larger, more scalable systems. Some of the larger systems have been extremely successful, others have been marginally to reasonably successful, and a few have been outright failures. Based on the author's experience programming the KSR1, Convex Exemplar, Silicon Graphics Inc. (SGI) Challenge and Power Challenge, and the SGI Origin 2000, some insights into key design issues for a successful *cache coherent shared memory symmetric multiprocessor* are discussed. The report concludes with a frequently overlooked issue—the cost effectiveness of some of these designs. In particular, any design that requires the widespread replication of key data structures will have a hard time establishing its cost effectiveness (even if it does meet the requirements for performance and scalability).

14. SUBJECT TERMS

supercomputer, high performance computing, parallel programming, SMP, symmetric multiprocessor

15. NUMBER OF PAGES

19

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2146 (Pressel) Date of Report January 2000

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

Organization

Name E-mail Name

Street or P.O. Box No.

City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

Organization

Name

Street or P.O. Box No.

City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)

(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
US ARMY RESEARCH LABORATORY
ATTN AMSRL CI HC
ABERDEEN PROVING GROUND MD 21005-5067